# 732A99 Lab Block 2

Rasmus Säfvenberg, Eleftheria Chatzitheodoridou

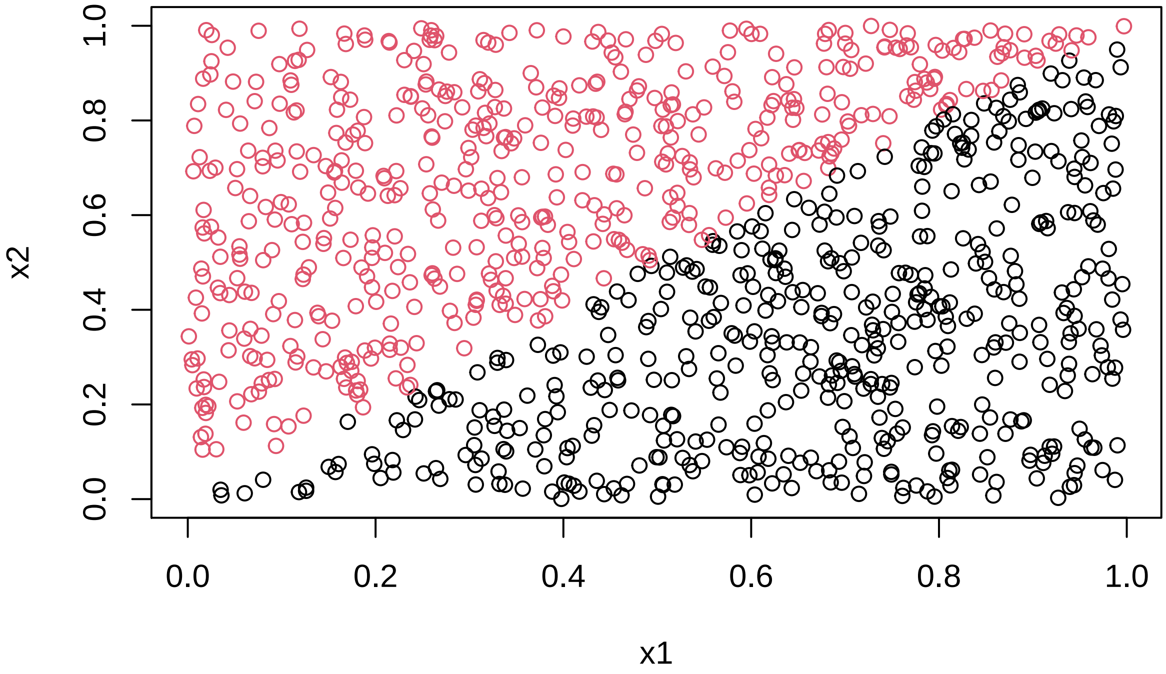
2020-12-02

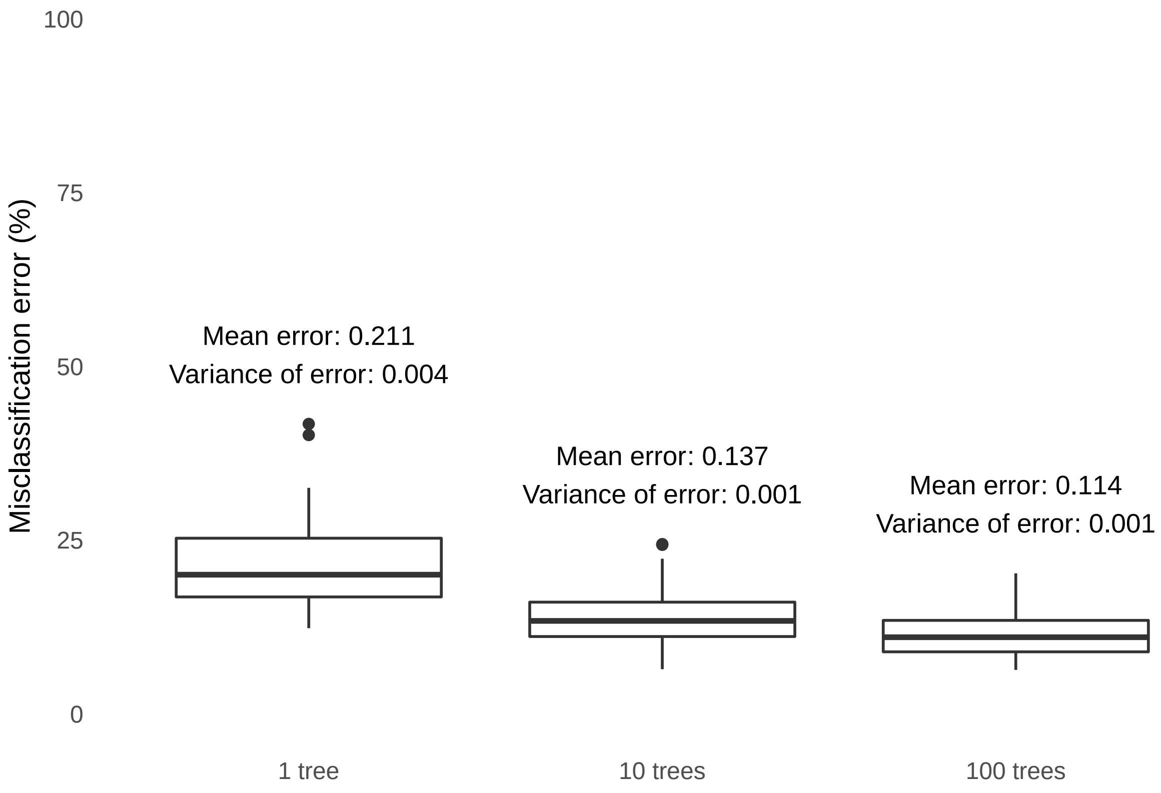
## Statement of Contribution

Rasmus produced Assignment 1 and Eleftheria was responsible for Assignment 3. Both parties discussed the results.

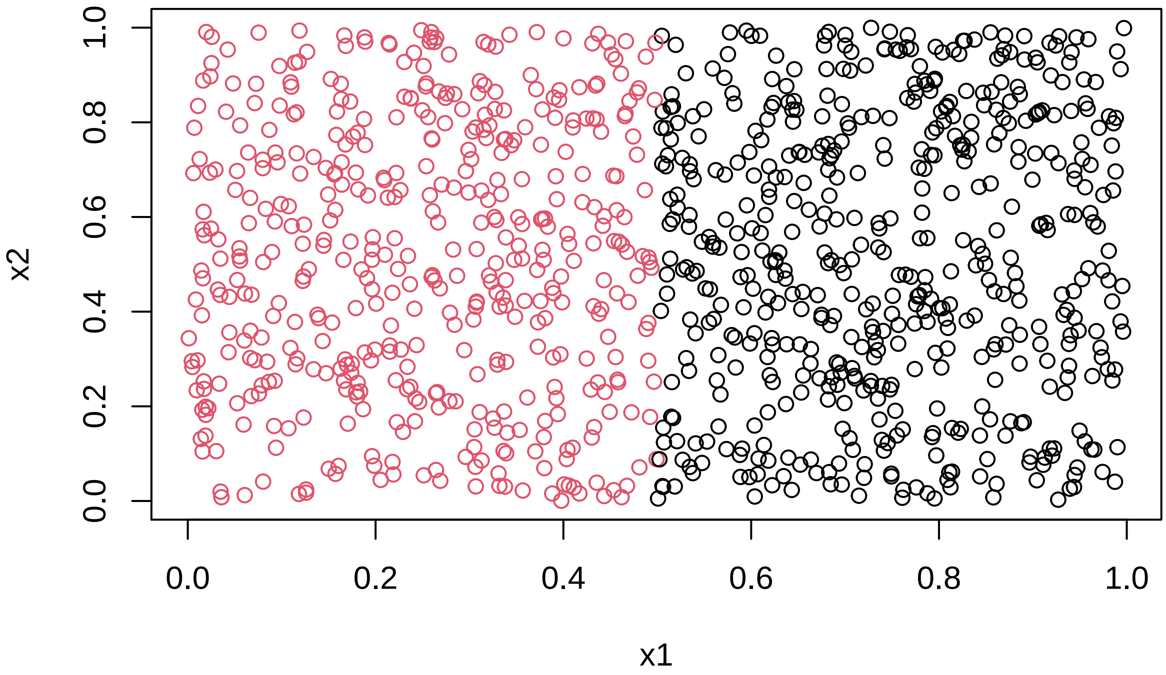
## Assignment 1: Ensemble Methods

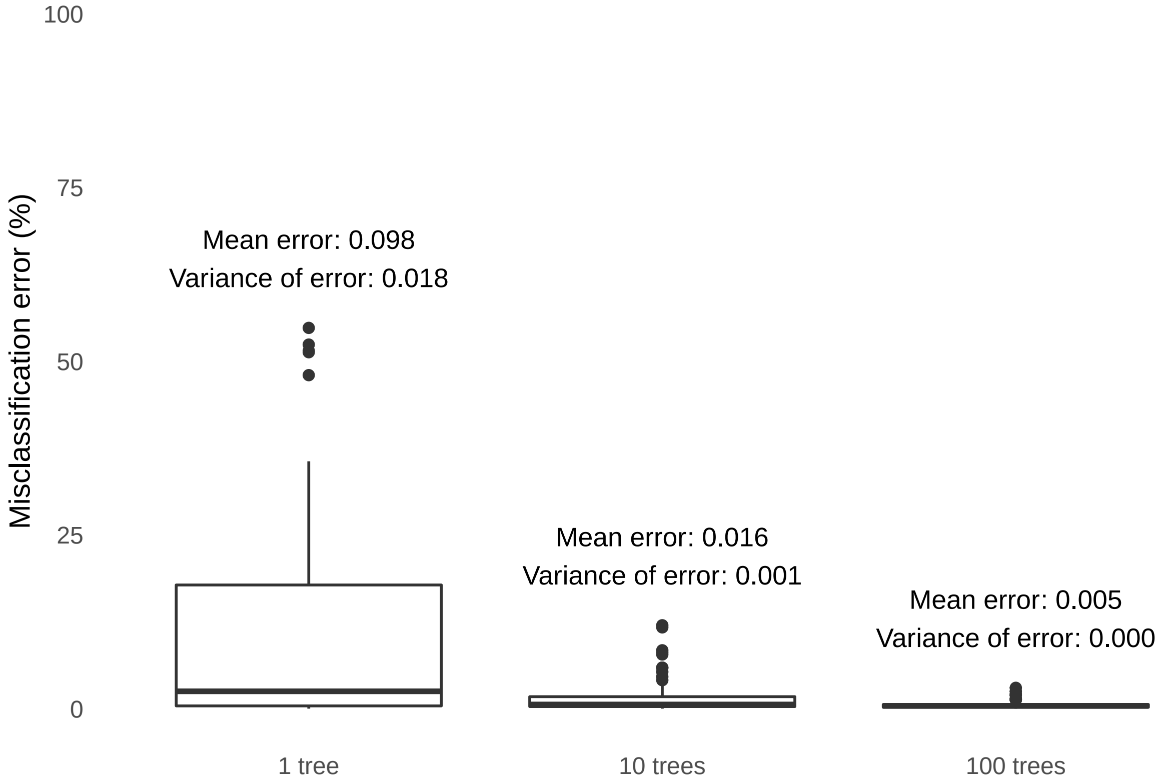
*1.a Repeat the procedure above for 100 training datasets and report the mean and variance of the misclassification errors. In other words, create 100 training datasets, learn a random forest from each dataset, and compute the misclassification error in the* **same** *test dataset. Report results for when the random forest has 1, 10 and 100 trees.*



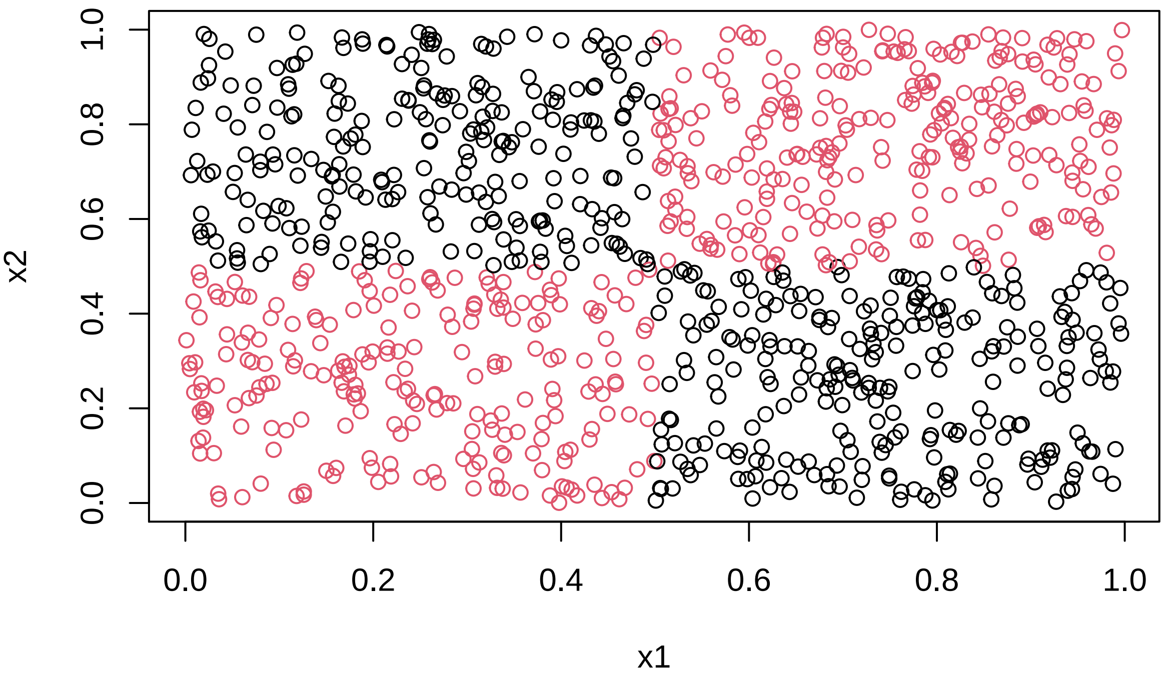


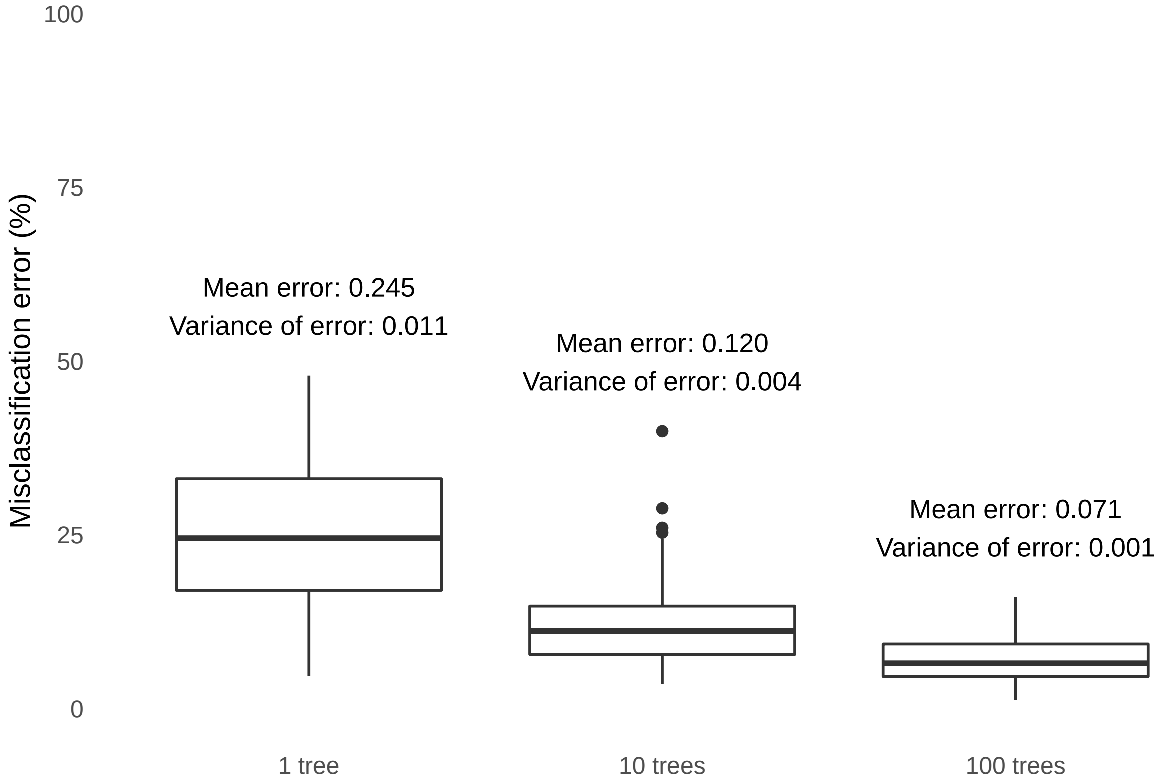
*1b. Repeat the exercise above but this time use the condition (`x1 < 0.5`) instead of (`x1 < x2`) when producing the training* **and** *test datasets.*


x1
x2
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0



*1c. Repeat the exercise above but this time use the condition* `((x1 < 0.5 & x2 < 0.5) | (x1 > 0.5 & x2 > 0.5))` *instead of* `(x1 < x2)` *when producing the training* **and** *test datasets. Unlike above, use* `nodesize = 12` *for this exercise.*


x2
x1
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0



**1d**

- a) *What happens with the mean and variance of the error rate when the number of trees in the random forest grows ?* Across all three scenarios, it can be seen that the mean error rate decreases as the number of trees increases, which can also be seen for the variance of the error, except in the first case, where the variance of 10 trees is slightly lower than that of 100 trees (6 decimal places). The reason the mean error rate is decreasing with increasing amount of trees can be explained by the majority voting principle, where more trees will, on average, predict better than an average individual tree. The same should also be applied to the variance to an extent, but the variance will eventually increase with larger forests as it overfits the data, which leads to poor generalizability on new data. This phenomenon is known as the bias-variance trade-off.

- b) *The third dataset represents a slightly more complicated classification problem than the first one. Still, you should get better performance for it when using sufficient trees in the random forest. Explain why you get better performance.* The first classification problem has, as observed from the graphs, a linear decision boundary with two distinct classes, which is a simple problem in nature. However, it can require many trees in a random forest to classify accurately due to the large amount of splits needed to partition the feature space into distinct classification regions, i.e., hypercubes. On the contrary, the third classification problem has four distinct regions(quadrants), of which the diagonal elements belong to the same class. This problem should have better performance than the aforementioned problem, since it requires less partitions of the feature space in order to achieve accurate classification.

- c) *Why is it desirable to have low error variance?* Having low error variance is desirable because it reduces the variability of the estimated error, thus leading to more accurate estimations around the mean. A high error variance can be thought of having bad aim in a game of darts; the darts end up all over the place, whereas low variance has the darts centered in a specific region.

## Assignment 2: Mixture Models

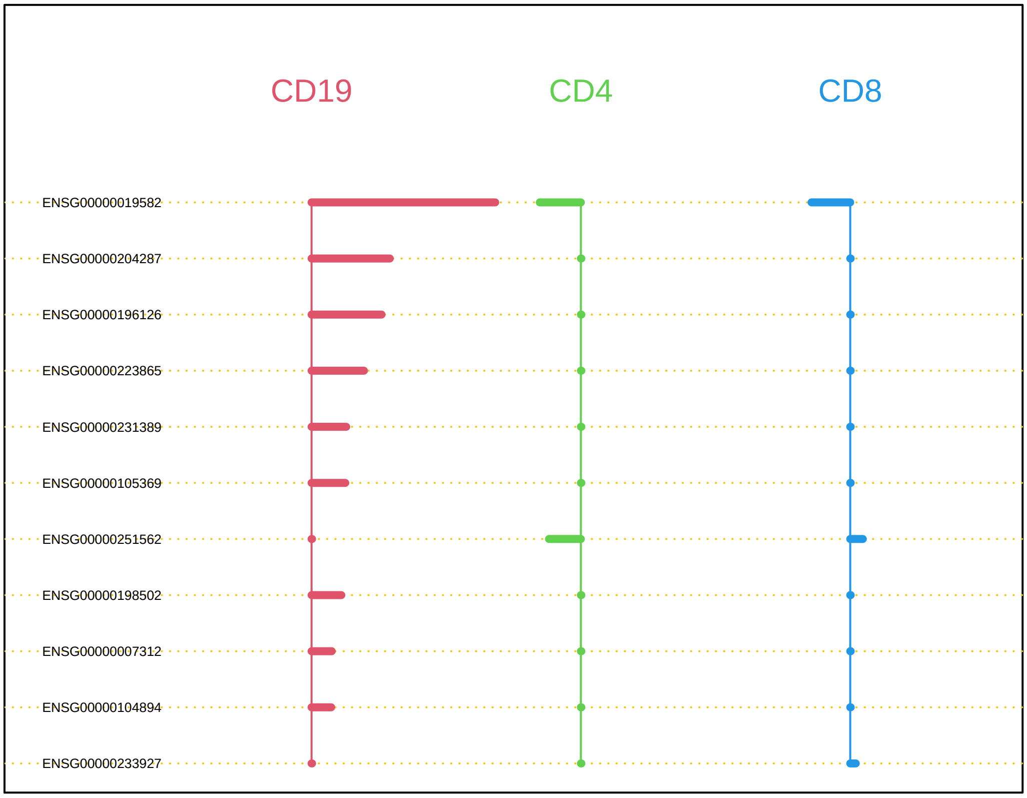
## Assignment 3: High-Dimensional Methods

*Data file* **geneexp.csv** *contains information about gene expression of three different cell types (column Cell Type). These cell types are CD4 and CD8 (two sorts of T cells) and CD19 (B cells). The aim of this assignment is to classify cells to the appropriate cell types using gene expressions and discover relevant genes for the given cell types.*

*1. Divide data into training and test sets (70/30) without scaling. Perform* **nearest shrunken centroid classification** *of training data in which the threshold is chosen by cross-validation. Provide a* **centroid plot** *and interpret it. How many genes were selected by the method? What meaning do positive and negative values have in the centroid plot? Can it happen that all values in the centroid plot are positive for some gene?*

```
## 123456789101112131415161718192021222324252627282930
```

```
## Call:
## pamr.cv(fit = model, data = mygene_train)
##    threshold nonzero errors
## 1   0.000    2058    24
## 2   0.801    1040    22
## 3   1.603     374    25
## 4   2.404     226    25
## 5   3.206     147    25
## 6   4.007     102    23
## 7   4.809      78    24
## 8   5.610      54    23
## 9   6.412      35    25
## 10  7.213      25    21
## 11  8.014      19    22
## 12  8.816      11    21
## 13  9.617      10    26
## 14 10.419       8    39
## 15 11.220       7    70
## 16 12.022       4    72
## 17 12.823       4    73
## 18 13.624       3    73
## 19 14.426       2    73
## 20 15.227       1    73
## 21 16.029       1    73
## 22 16.830       1    72
## 23 17.632       1    72
## 24 18.433       1    72
## 25 19.235       1    72
## 26 20.036       1    71
## 27 20.837       1    70
## 28 21.639       1    70
## 29 22.440       1    92
## 30 23.242       0    118
```



```
##       id  CD19-score CD4-score CD8-score
##  [1,] 2   1.3637     -0.3053   -0.2873
##  [2,] 15  0.5812     0         0
##  [3,] 31  0.52       0         0
##  [4,] 32  0.3888     0         0
##  [5,] 37  0.2574     0         0
##  [6,] 138 0.2504     0         0
##  [7,] 1   0          -0.2369   0.0937
##  [8,] 90  0.2212     0         0
##  [9,] 172 0.15       0         0
## [10,] 126 0.1459     0         0
## [11,] 79  0          0         0.0412
```

For this task we are asked to perform Nearest Shrunken Centroid Classification of training, non-scaled data in which the threshold is chosen by cross- validation. After the execution of the method, we choose the threshold that yields the minimum misclassification error. Since there are 2 thresholds satisfying this condition, we pick the one that includes the least amount of variables, which equals to ≈ 8.8158. Essentially, this method improves the accuracy of the classifier by reducing the effect of noisy genes and performs automatic gene selection. In particular, "if a gene is shrunk to zero for all classes, then it is eliminated from the prediction rule. Alternatively, it may be set to zero for all classes except one, and we learn that high or low expression for that gene characterizes that class" [1].

The centroid plot shows the distance of each feature/gene from the three classes (CD4, CD8, CD19). Each centroid has the overall centroid subtracted; hence, what we see are contrasts. The red column corresponds to class CD19, the green one to CD4 and the blue one to CD8. Moreover, a centroid plot reports which non-zero variables are the most significant ones for the classifier. In our plot, we see 11 of them selected for the optimal threshold.

Positive and negative values indicate gene expression. More specifically, a positive value implies a strong gene expression for a particular cell type and a positive contribution to the classification. It is our opinion that the values of a gene cannot be all positive. This would translate to all the values of this gene being far from the centroids. In a mathematical context, when the algorithm reaches the shrinkage step, that is the threshold reduction towards the mean, it would render the contribution of this gene to 0, thus eliminating it from the centroid plot.

*2. List the names of the 2 most contributing genes and find their alternative names in Google. Then, by checking this webpage https://panglaodb.se/markers.html find out whether these two genes are "marker genes" for given cell types. Report the test error of the model.*

```
##      [,1]
## [1,] "ENSG00000019582"
## [2,] "ENSG00000204287"
```

```
## [1] 0.1444444
```

**The two most contributing genes**

1. "ENSG00000019582" also known as CD74 molecule (DHLAG)
2. "ENSG00000204287" also known as HLA-DRA (HLA-DRA1)

Both of these genes belong to B cells therefore to the CD19 cell type and are "marker genes" for these cells. The second (HLA-DRA) can be found only in humans, whilst the first (CD74) can be encountered both in humans and mice. In general, B cells are a type of white blood cell and their functionality is of autoimmune nature by secreting antibodies. [4]

The test error of the model is equal to: 0.1444444

*3. Compute the test error and the number of the contributing features for the following methods fitted to the training data:*

a. *Elastic net with the binomial response and* $\alpha = 0.5$ *in which penalty is selected by the cross-validation.*
b. *Support vector machine with "vanilladot" kernel.*

*Compare the results of these models with the results of the nearest shrunken centroids (make a comparative table). Which model would you prefer and why?*

```
##                    Nearest.Shrunken.Centroid  Elastic.net         SVM
## Selected Features                 11.0000000 144.00000000 74.00000000
## Test Error                         0.1444444   0.05555556  0.02222222
```

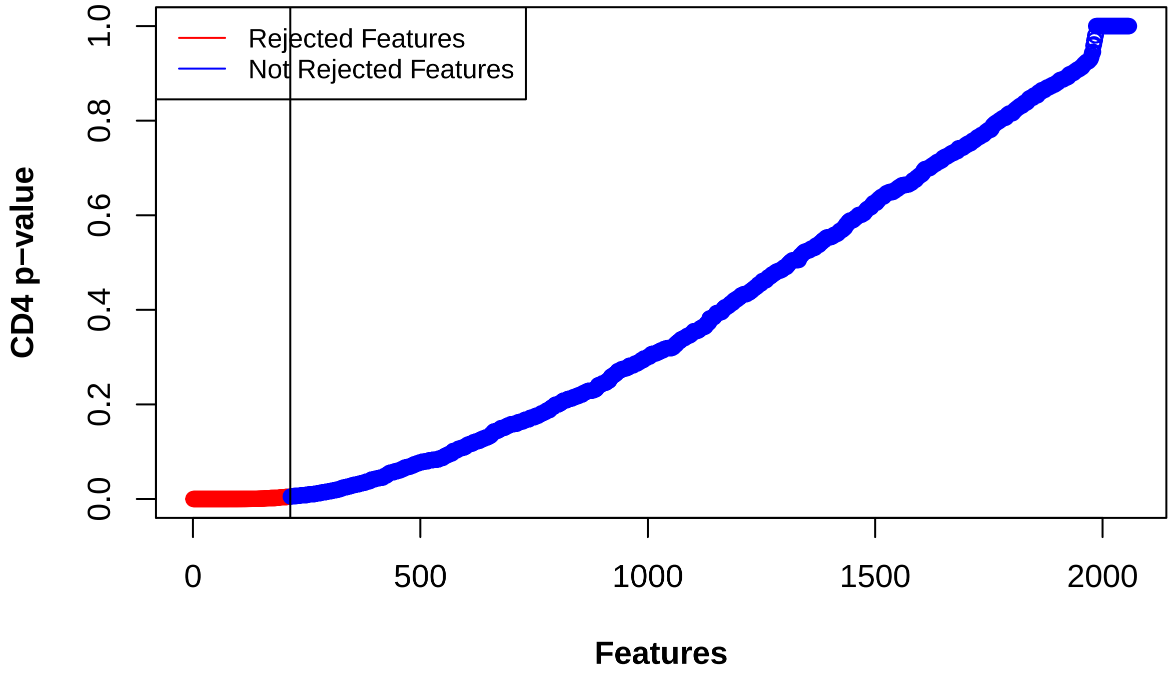
The model with the lowest misclassification error rate is the Support Vector Machine (SVM) one between the 3 methods. But in order to choose which model performed best we would also need to look at the number of features that were selected. Elastic Net and SVM have the highest amount of variables selected which makes them inefficient as they render complex models. Therefore, our choice is automatically limited to SVM and NSC. Having taken both criteria into consideration, we would choose SVM because, even though the number of features is not the lowest, the difference between 144 and 74 variables is significant and it does still have the lowest test error.

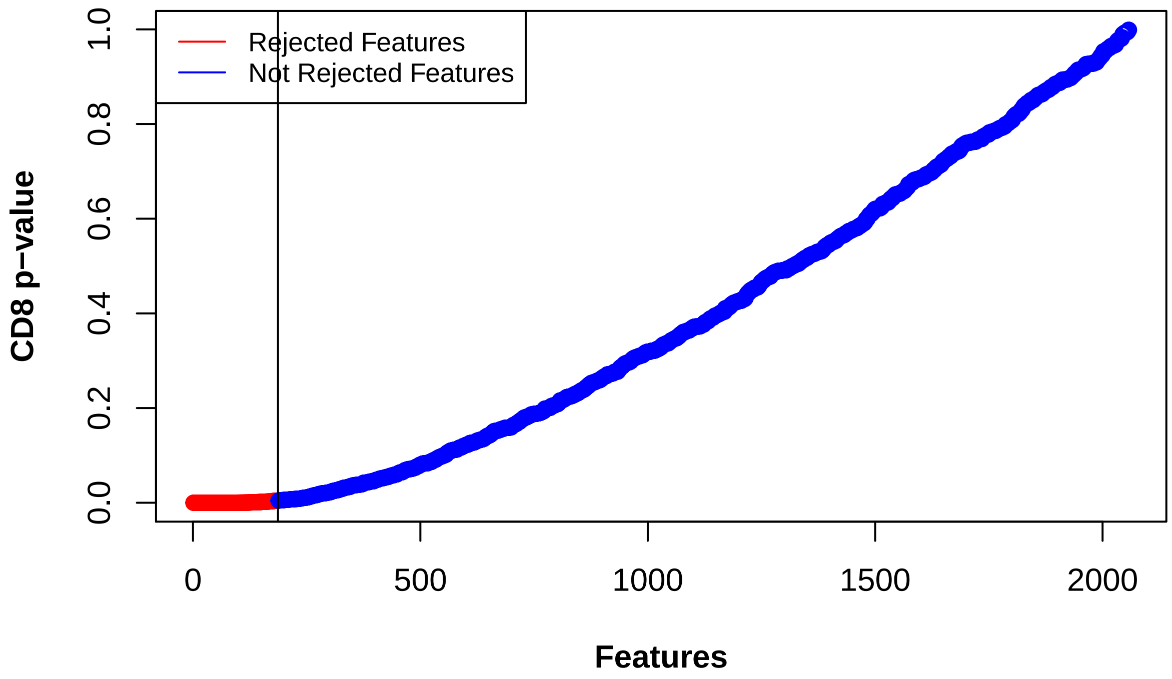
*4. Implement Benjamini-Hochberg method for the original data in which you test each cell type versus the remaining ones, and use `t.test()` for computing p-values. Present plots showing p-values and the rejection area for each cell type and interpret them. How many genes correspond to the rejected hypotheses for each cell type?*

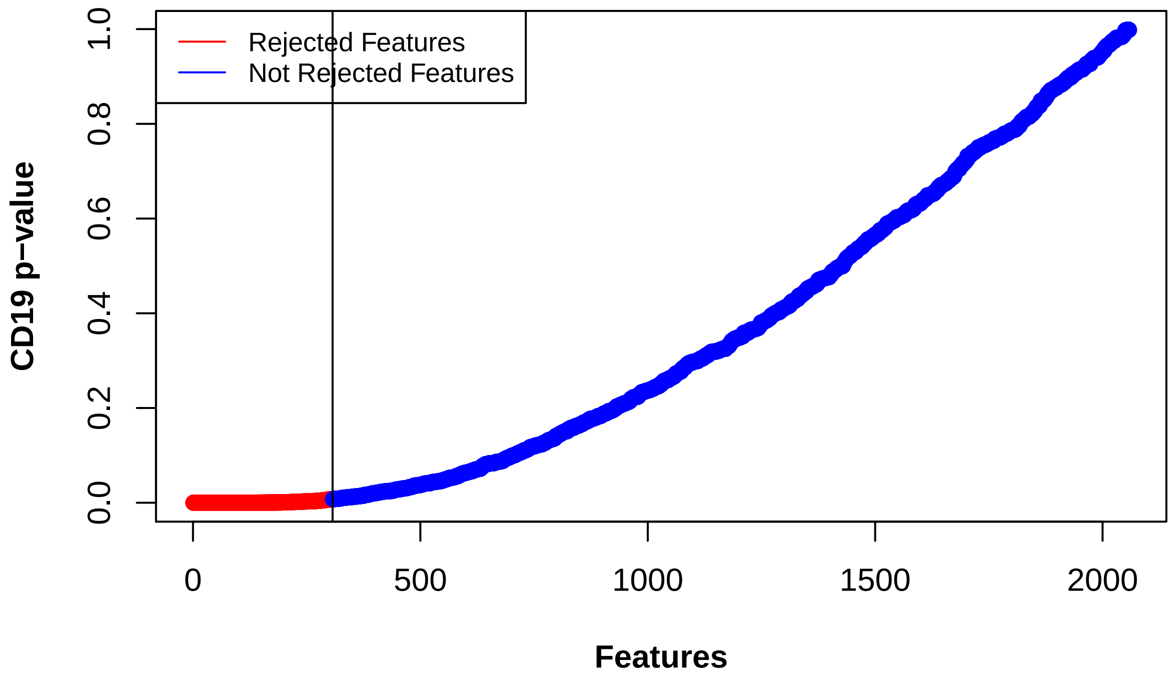
```
## [1] 214
```

```
## [1] 187
```

```
## [1] 307
```




Rejected Features
Not Rejected Features
CD8 p−value
0.0
0.2
0.4
0.6
0.8
1.0
0
500
1000
1500
2000
Features



For this task, we were asked to implement the Benjamini-Hochberg method for the original training data. This algorithm helps us lower the False Discovery Rate (FDR). More specifically, let as assume the following hypotheses:

$H_{0j}$ = feature (gene) has no effect on the classification of the cell type

$H_{1j}$ = feature (gene) has an effect on the classification of the cell type

After calculating p-values for each cell type using `t.test()` we keep the values that are $\leq 0.05$ (arbitrary choice) using the rejection formula

$$L = max\{j : p(j) \leq a * j/M\}$$

The aforementioned formula returns the Benjamini-Hochberg threshold. This means that the Hypothesis Test will be rejected only for the features whose p-values are less than or equal to the p-value of the threshold. For the CD4 cell type 214 features were rejected which means that those features are significant for the classification of the cell type. Similarly, 187 features were rejected for the CD8 cell type and 307 for the CD19 cell type.

For the plots, we draw p-values for the three cell types (CD4, CD8, CD19) vs. the total number of features. Rejected features are represented by red color. All three plots show an upward trend which is proportionate to the increase of the number of features. In other words, as the number of features increases, so does the p-value. The number of the rejected features does not exceed 310.

## References

1. Statweb. What is nearest shrunken centroid classification? [2020]

[http://statweb.stanford.edu/~tibs/PAM/Rdist/howwork.html]

2. Research Gate. Nearest Shrunken Centroid as Feature Selection of Microarray Data [2020]

[https://www.researchgate.net/publication/221206450_Nearest_Shrunken_Centroid_as_Feature_Selection_of_Microarray_Data]

3. Datacamp. Regularization:Ridge, Lasso & Elastic Net [2020]

[https://www.datacamp.com/community/tutorials/tutorial-ridge-lasso-elastic-net]

4. Panglaodb. Cell type gene expression markers [2020]

[https://panglaodb.se/markers.html?cell_type=%27B%20cells%27]

5. e!Ensembl. Gene: HLA-DRA [2020]

[http://www.ensembl.org/Homo_sapiens/Gene/Summary?g=ENSG00000204287;r=6:32439878-32445046]

6. e!Ensembl. Gene: CD74 [2020]

[http://www.ensembl.org/Homo_sapiens/Gene/Summary?g=ENSG00000019582;r=5:150400041-150412929]

## Appendix

```r
## Setup ##
knitr::opts_chunk$set(echo = FALSE, warning = FALSE, message = FALSE,
                      out.width = "100%")
library(randomForest)
library(ggplot2)
library(reshape2)
library(dplyr)
library(pamr)
library(glmnet)
library(kernlab)

## Assignment 1: Ensemble Methods

# 1.a
# Test
set.seed(12345)

x1 <- runif(1000)
x2 <- runif(1000)
tedata <- cbind(x1, x2)
y <- as.numeric(x1 < x2)
telabels <- as.factor(y)
plot(x1, x2, col = (y + 1))
```

```r
# Train
misclass_1 <- vector("numeric", length = 100L)
misclass_10 <- vector("numeric", length = 100L)
misclass_100 <- vector("numeric", length = 100L)
set.seed(12345)
for(i in 1:100){
  x1 <- runif(100)
  x2 <- runif(100)
  trdata <- cbind(x1, x2)
  y <- as.numeric(x1 < x2)
  trlabels <- as.factor(y)
  for(num_trees in c(1, 10, 100)) {
    forest <- randomForest(trlabels ~ x1 + x2, ntree = num_trees, nodesize = 25,
                           keep.forest = TRUE)
    pred <- predict(forest, tedata)
    if(num_trees == 1) {
      misclass_1[i] <- (sum(telabels != pred)) / length(telabels)
    } else if(num_trees == 10) {
      misclass_10[i] <- (sum(telabels != pred)) / length(telabels)
    } else {
      misclass_100[i] <- (sum(telabels != pred)) / length(telabels)
    }
  }
}

df <- melt(data.frame(misclass_1, misclass_10, misclass_100))

df %>%
  group_by(variable) %>%
  mutate(mean = mean(value), variance = var(value)) %>%
  ungroup() %>%
  ggplot(aes(variable, value * 100)) + geom_boxplot() +
  xlab("") + ylab("Misclassification error (%)") +
  scale_x_discrete(labels = c("1 tree", "10 trees", "100 trees")) +
  scale_y_continuous(limits = c(0, 100)) + theme_minimal() +
  theme(panel.grid = element_blank()) +
  stat_summary(geom = "text", fun = max,
               aes(label = paste0("Mean error: ", sprintf("%1.3f", mean), "\n",
                                  "Variance of error: ", sprintf("%1.3f", variance))),
               position = position_nudge(y = 10), size = 3.5)

# 1.b
# Test
set.seed(12345)

x1 <- runif(1000)
x2 <- runif(1000)
tedata <- cbind(x1, x2)
y <- as.numeric(x1 < 0.5)
telabels <- as.factor(y)
plot(x1, x2, col = (y + 1))

# Train
```

```r
misclass_1 <- vector("numeric", length = 100L)
misclass_10 <- vector("numeric", length = 100L)
misclass_100 <- vector("numeric", length = 100L)
set.seed(12345)
for(i in 1:100){
  x1 <- runif(100)
  x2 <- runif(100)
  trdata <- cbind(x1, x2)
  y <- as.numeric(x1 < 0.5)
  trlabels <- as.factor(y)
  for(num_trees in c(1, 10, 100)){
    forest <- randomForest(trlabels ~ x1 + x2, ntree = num_trees, nodesize = 25,
                           keep.forest = TRUE)
    pred <- predict(forest, tedata)
    if(num_trees == 1){
      misclass_1[i] <- (sum(telabels != pred)) / length(telabels)
    } else if(num_trees == 10){
      misclass_10[i] <- (sum(telabels != pred)) / length(telabels)
    } else{
      misclass_100[i] <- (sum(telabels != pred)) / length(telabels)
    }
  }
}

df <- melt(data.frame(misclass_1, misclass_10, misclass_100))
df %>%
  group_by(variable) %>%
  mutate(mean = mean(value), variance = var(value)) %>%
  ungroup() %>%
  ggplot(aes(variable, value * 100)) + geom_boxplot() +
  xlab("") + ylab("Misclassification error (%)") +
  scale_x_discrete(labels = c("1 tree", "10 trees", "100 trees")) +
  scale_y_continuous(limits = c(0, 100)) + theme_minimal() +
  theme(panel.grid = element_blank()) +
  stat_summary(geom = "text", fun = max,
               aes(label = paste0("Mean error: ", sprintf("%1.3f", mean), "\n",
                                  "Variance of error: ", sprintf("%1.3f", variance))),
               position = position_nudge(y = 10), size = 3.5)

# 1.c
# Test
set.seed(12345)

x1 <- runif(1000)
x2 <- runif(1000)
tedata <- cbind(x1, x2)
y <- as.numeric(((x1 < 0.5 & x2 < 0.5) | (x1 > 0.5 & x2 > 0.5)) )
telabels <- as.factor(y)
plot(x1, x2, col = (y + 1))

# Train
misclass_1 <- vector("numeric", length = 100L)
misclass_10 <- vector("numeric", length = 100L)
```

```
misclass_100 <- vector("numeric", length = 100L)
set.seed(12345)
for(i in 1:100){
  x1 <- runif(100)
  x2 <- runif(100)
  trdata <- cbind(x1, x2)
  y <- as.numeric(((x1 < 0.5 & x2 < 0.5) | (x1 > 0.5 & x2 > 0.5)) )
  trlabels <- as.factor(y)
  for(num_trees in c(1, 10, 100)){
    forest <- randomForest(trlabels ~ x1 + x2, ntree = num_trees, nodesize = 12,
                           keep.forest = TRUE)
    pred <- predict(forest, tedata)
    if(num_trees == 1){
      misclass_1[i] <- (sum(telabels != pred)) / length(telabels)
    } else if(num_trees == 10){
      misclass_10[i] <- (sum(telabels != pred)) / length(telabels)
    } else{
      misclass_100[i] <- (sum(telabels != pred)) / length(telabels)
    }
  }
}

df <- melt(data.frame(misclass_1, misclass_10, misclass_100))
df %>%
  group_by(variable) %>%
  mutate(mean = mean(value), variance = var(value)) %>%
  ungroup() %>%
  ggplot(aes(variable, value * 100)) + geom_boxplot() +
  xlab("") + ylab("Misclassification error (%)") +
  scale_x_discrete(labels = c("1 tree", "10 trees", "100 trees")) +
  scale_y_continuous(limits = c(0, 100)) + theme_minimal() +
  theme(panel.grid = element_blank()) +
  stat_summary(geom = "text", fun = max,
               aes(label = paste0("Mean error: ", sprintf("%1.3f", mean), "\n",
                                  "Variance of error: ", sprintf("%1.3f", variance))),
               position = position_nudge(y = 10), size = 3.5)


# 1.d
no code


## Assignment 2: Mixture Models
# Template
set.seed(12345)

max_it <- 100 # max number of EM iterations
min_change <- 0.1 # min change in log likelihood between two consecutive EM iterations
N <- 1000 # number of training points
D <- 10 # number of dimensions
x <- matrix(nrow = N, ncol = D) # training data

true_pi <- vector(length = 3) # true mixing coefficients
```

```
true_mu <- matrix(nrow = 3, ncol = D) # true conditional distributions
true_pi <- c(1/3, 1/3, 1/3)
true_mu[1,] <- c(0.5, 0.6, 0.4, 0.7, 0.3, 0.8, 0.2, 0.9, 0.1, 1)
true_mu[2,] <- c(0.5, 0.4, 0.6, 0.3, 0.7, 0.2, 0.8, 0.1, 0.9, 0)
true_mu[3,] <- c(0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5)
plot(true_mu[1,], type = "o", col = "blue", ylim = c(0,1))
points(true_mu[2,], type = "o", col = "red")
points(true_mu[3,], type = "o", col = "green")

# Producing the training data
for(n in 1:N) {
k <- sample(1:3, 1, prob = true_pi)
for(d in 1:D) {
x[n,d] <- rbinom(1, 1, true_mu[k,d])
 }
}

K <- 3 # number of guessed components
z <- matrix(nrow = N, ncol = K) # fractional component assignments
pi <- vector(length = K) # mixing coefficients
mu <- matrix(nrow = K, ncol = D) # conditional distributions
llik <- vector(length = max_it) # log likelihood of the EM iterations

# Random initialization of the paramters
pi <- runif(K, 0.49, 0.51)
pi <- pi / sum(pi)
for(k in 1:K) {
mu[k,] <- runif(D, 0.49, 0.51)
}
pi
mu

for(it in 1:max_it) {
plot(mu[1,], type = "o", col = "blue", ylim = c(0,1))
points(mu[2,], type = "o", col = "red")
points(mu[3,], type = "o", col = "green")
#points(mu[4,], type = "o", col = "yellow")
Sys.sleep(0.5)

# E-step: Computation of the fractional component assignments
# Your code here

# Log likelihood computation.
# Your code here

cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
flush.console()
# Stop if the lok likelihood has not changed significantly
# Your code here

#M-step: ML parameter estimation from the data and fractional component assignments
# Your code here
}
```

```
pi

mu
plot(llik[1:it], type = "o")


## Assignment 3: High-Dimensional Methods

# 3.1
# Read data
gene <- read.csv("geneexp.csv", header =  TRUE, row.names = 1, sep = ",")

# Divide data into training and test(70/30, no scaling)
n <- dim(gene)[1]
set.seed(12345)
id <- sample(1:n, floor(n*0.7))
train <- gene[id,]
test <- gene[-id,]

# train
rownames(train) <- 1:nrow(train)
x_train <- t(train[, -2086]) # remove last column
y_train <- train[[2086]]
mygene_train <- list(x = x_train, y = as.factor(y_train), geneid = as.character(1:nrow(x_train)), genena

# test
rownames(test) <- 1:nrow(test)
x_test <- t(test[,-2086])
y_test <- test[[2086]]
mygene_test <- list(x = x_test, y = as.factor(y_test), geneid = as.character(1:nrow(x_test)), genenames

# Create model
model <- pamr.train(mygene_train)

# Choose threshold by cross-validation
invisible(capture.output(cvmodel <- pamr.cv(model, mygene_train)))

# Print cvmodel
print(cvmodel)

# Optimal threshold is the one that gives the minimum cross-validated misclassification error rate

optimal_threshold <- cvmodel$threshold[which(cvmodel$error == min(cvmodel$error))[-1]]

# Centroid plot
pamr.plotcen(model, mygene_train, threshold = optimal_threshold)

# Number of features:
features <- pamr.listgenes(model, mygene_train, threshold = optimal_threshold) # 11
n_features <- nrow(features)

# 3.2
# List 2 most contributing genes
```

```r
best_2 <- as.matrix(colnames(gene)[as.numeric(features[,1])][1:2])
best_2 # "ENSG00000019582", "ENSG00000204287"

# Calculate test error
pred_NSC <- pamr.predict(model, newx = x_test, threshold = optimal_threshold)
conf_pred_model <- table(y_test, pred_NSC)
#conf_pred_model
test_error_NSC <- 1 - sum(diag(conf_pred_model)) / sum(conf_pred_model)
test_error_NSC

# 3.3
# Elastic net

set.seed(12345)

elastic_net <- cv.glmnet(x = t(x_train), y = y_train, family = "multinomial", alpha = 0.5) #fitting the

pred_elastic_net <- predict(elastic_net, newx = t(x_test), type = "class", s = "lambda.min")

# Calculate test error
conf_elastic_net  <- table(y_test, pred_elastic_net)
test_error_EN <- 1- sum(diag(conf_elastic_net)) / sum(conf_elastic_net)

# Support Vector Machine with "vanilladot" kernel

invisible(capture.output(SVM  <- ksvm(x = t(x_train), y = as.factor(y_train),  kernel = "vanilladot", s

SVM_pred <- predict(SVM, newdata = t(x_test))

# Calculate test error
conf_SVM <- table(y_test, SVM_pred)
test_error_SVM <- 1 - sum(diag(conf_SVM)) / sum(conf_SVM)

# Compare the 3 methods by creating a table with the values of the test error
df <- data.frame("Nearest Shrunken Centroid" = test_error_NSC,
                 "Elastic net" = test_error_EN,                      # getting test errors for every met
                  "SVM" = test_error_SVM)

cf <- coef(elastic_net, elastic_net$lambda.min)
cf <- as.matrix(cbind(cf[[1]], cf[[2]], cf[[3]]))
cf2 <- unique(which(cf != 0, arr.ind = T)[,1])

features_NSC <- n_features
features_SVM  <- SVM@nSV              #length(coef(SVM)[[1]])        # getting all features for every me
features_EN <- length(cf2)


features <- c(features_NSC, features_EN, features_SVM)               # combine features+test errors in a
df <- rbind(features, df)
rownames(df) <- c("Selected Features", "Test Error")

df
```

```
# 3.4
# Implement Benjamini-Hochberg for original data

gene_BH <- train
gene_BH$CD4 <- ifelse(gene_BH$CellType == "CD4", 1, 0)
gene_BH$CD8 <- ifelse(gene_BH$CellType == "CD8", 1, 0)
gene_BH$CD19 <- ifelse(gene_BH$CellType == "CD19", 1, 0)

# Store p-values for each cell type
CD4_pvalue <- apply(gene_BH[, -c(2086:2089)], 2, function(x) t.test(x ~ gene_BH$CD4)$p.value)
names(CD4_pvalue) <- colnames(train)[-2086]
CD4_pvalue <- sort(na.omit(CD4_pvalue)) # remove NaNs

CD8_pvalue <- apply(gene_BH[, -c(2086:2089)], 2, function(x) t.test(x ~ gene_BH$CD8)$p.value)
names(CD8_pvalue) <- colnames(train)[-2086]
CD8_pvalue <- sort(na.omit(CD8_pvalue))

CD19_pvalue <- apply(gene_BH[, -c(2086:2089)], 2, function(x) t.test(x ~ gene_BH$CD19)$p.value)
names(CD19_pvalue) <- colnames(train)[-2086]
CD19_pvalue <- sort(na.omit(CD19_pvalue))

alpha <- 0.05
l <- alpha * (1:length(CD4_pvalue) / length(CD4_pvalue))
L <- max(l)

BH_method <- function(p_values) {
  rej_lim <- 0
  M <- length(p_values)
  for(j in 1:M) {
    if(p_values[j] <= (alpha * j / M)) {
      rej_lim <- j
    }
  }
  rej_lim
}

BH_method(CD4_pvalue)
BH_method(CD8_pvalue)
BH_method(CD19_pvalue)

col4 <- ifelse(CD4_pvalue <= CD4_pvalue[BH_method(CD4_pvalue)], "red", "blue")
CD4_rej <- which(col4 == "red") # rejected features for CD4 cell type, 214
col8 <- ifelse(CD8_pvalue <= CD8_pvalue[BH_method(CD8_pvalue)], "red", "blue")
CD8_rej <- which(col8 == "red") # rejected features for CD8 cell type, 187
col19 <- ifelse(CD19_pvalue <= CD19_pvalue[BH_method(CD19_pvalue)], "red", "blue")
CD19_rej <- which(col19 == "red") # rejected features for CD19 cell type, 307

plot(CD4_pvalue, col = col4, xlab = "Features", ylab = "CD4 p-value", font.lab = 2)
abline(abline(v = BH_method(CD4_pvalue), col = "black"))
legend("topleft", legend = c("Rejected Features", "Not Rejected Features"), col = c("red", "blue"), lwd

plot(CD8_pvalue, col = col8, xlab = "Features", ylab = "CD8 p-value", font.lab = 2)
abline(abline(v = BH_method(CD8_pvalue), col = "black"))
```

```
legend("topleft", legend = c("Rejected Features", "Not Rejected Features"), col = c("red", "blue"), lwd

plot(CD19_pvalue, col= col19, xlab = "Features", ylab = "CD19 p-value", font.lab = 2)
abline(abline(v = BH_method(CD19_pvalue), col = "black"))
legend("topleft", legend = c("Rejected Features", "Not Rejected Features"), col = c("red", "blue"), lwd
```